Project 3: Long-Short Term Memory Networks

Urszula Białończyk, Olaf Werner

May 2021

Contents

1 Introduction

The aim of the project was to build a model which would be a solution to TensorFlow Speech Recognition Challenge [3]. The competition's goal was to train a model which would recognize what word was said on 1-second recordings. We focused on two approaches: long-short term memory networks that are known to be effective in speech recognition tasks and convolutional neural networks that, after a proper data conversion, can also solve the given problem. We tried out different architectures and data augmentation so as to check what improves the models performance and what causes that the models learn less effectively.

2 Data set description

The training dataset consists of 30 different subfolders with 1 second clips of voice commands (the clips are in .wav format), with the folder name being the label of the audio clip. The labels we needed to predict in the test set are:

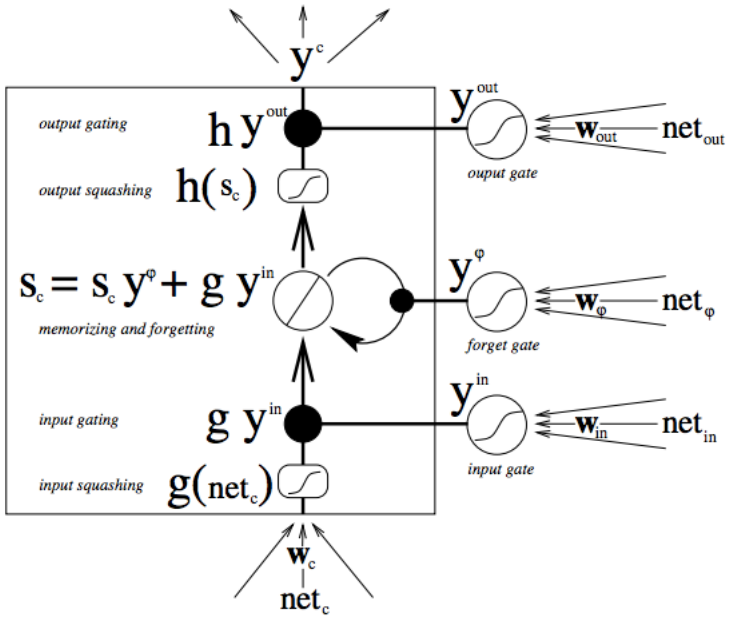
- yes,
- no,
- up,
- down,
- left,
- right,
- on,
- off,
- stop,
- go.

This means that there are more labels that should be predicted. We incorporated to the dataset the additional subfolders with their labels set to "unknown". Additionally, there's a folder called _background_noise_ which contains longer clips of "silence" that can be broke up and used as training input. At first we needed to preprocess the dataset in order to feed the data to the network. We decided to convert the .wav files to spectrograms. Additionally, since the dataset converted in such way was large (around 2GB), we decided to change the created images from RGB to grayscale. We didn't lose any important information by such conversion and the size of the dataset decreased to 600Mb. The testing dataset contains over 150 000 unlabelled observations. We preprocessed it in the same way as the training set.

3 Theoretical part

3.1 LSTMs

LSTMs [1] are class of neural networks designed to process time series data. It achieves it by using states at previous time steps to predict current states.



Architecture of LSTM unit. Image taken from [2]

Image 3.1 is an basic architecture of single LSTM unit. The sets of 3 arrows represents information from past output of the cell and current input. $w_c, w_\phi, w_{in}, w_{out}$ are weights. S_c is state of the cell it is kept between iterations, but can be modified. White circles and white opaque rectangles are activation functions. Now we will describe information flow. Starting from the bottom we calculate parameter g. It is used to scale y^{in}. Then we calculate y^{in} using input gate and scale it by g. After that we calculate y^ϕ using forget gate. We modify S_c using this formula $S_c = S_c y^\phi + g y^{in}$. Using new S_c we calculate parameter h. This parameter is used to scale y^{out}. We calculate y^{out} using output gate scale it by h and create new output y^c.

3.2 CNNs

Since we converted the .wav files to spectrograms, one of the possible solutions was to build convolutional neural networks, the basic approach to image recognition problem. But what is different from the task of the second project, where we built CNNs on Cifar10 dataset, is that we now have time information in our

images which is crucial to determine which word was spoken. This is important especially when it comes to data augmentation since the standard procedures can mask the effect of time in our data.

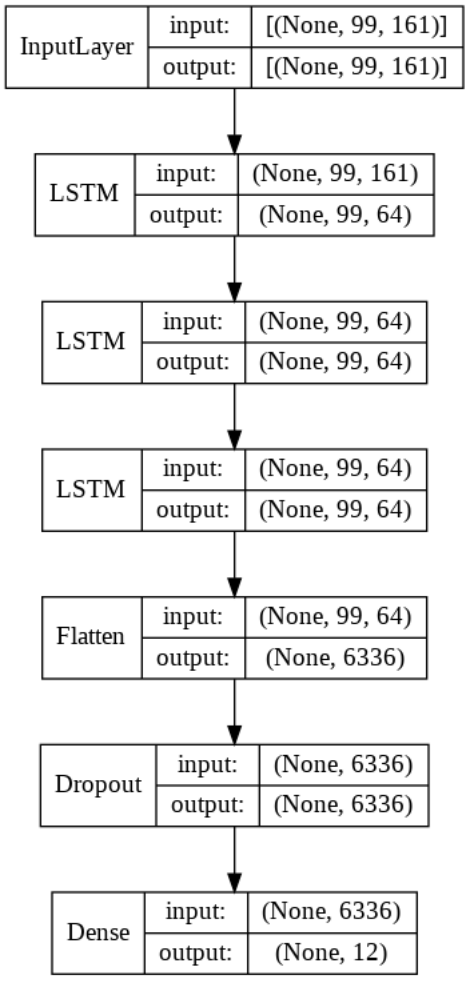
4 Experiments

4.1 Background noise preprocessing

Data in the _background_noise_ folder was splitted into 1 second clips. Than we performed following data augmentation on each of the following clips.

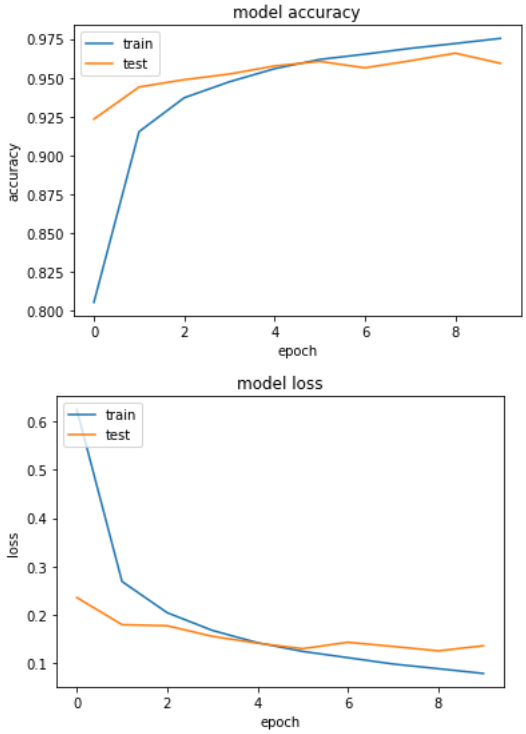
- up to 10% shift left or right
- up to 10% shift up or down
- vertical flip
- horizontal flip
- up to 20% brightness shift
- up to 10 degrees rotation

4.2 LSTMs



Architecture of our LSTM network.

Our architecture image 4.2 is stack of three LSTM layers. Every Layer is made of 64 LSTM units. At the end we flatten output of LSTM and add dropout of 0.3. The last layer is output layer made of 12 units, one for each class.



Training and Validation loss and accuracy.

As we can see in the image 4.2 we achieved very good results of 95% on Validation. There is no significant overfitting. However when we uploaded our predictions to Kaggle (image 4.2 we achieved much worse result of 74%.

Name	Submitted	Wait time	Execution time	Score
results(2).csv	14 hours ago	1 seconds	1 seconds	0.74396

Complete

Jump to your position on the leaderboard

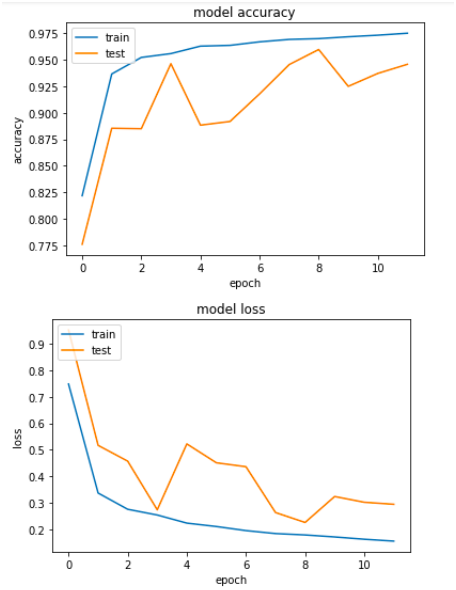
Training and Validation loss and accuracy.

4.3 CNNs

We decided to use ResNets to solve the task. We chose this architecture for two reasons: firstly, it generally performs well in image recognition problems. Secondly, we read that it can be a good choice when it comes to time series problems and one of the dimensions of our input data can be viewed as a time column.
We tested how the depth of the network and data augmentation influence the model's performance.

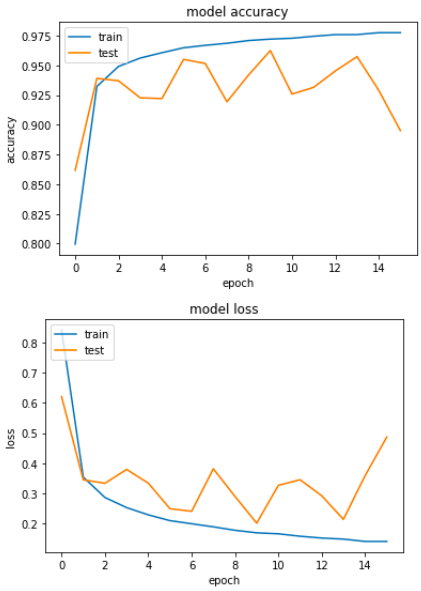
4.3.1 The depth of the network

We decided to test 1-, 2- and 3-layer ResNet architectures. Below we present the results obtained for training and testing sets (the test set was created manually, we simply split the dataset into training and testing parts).



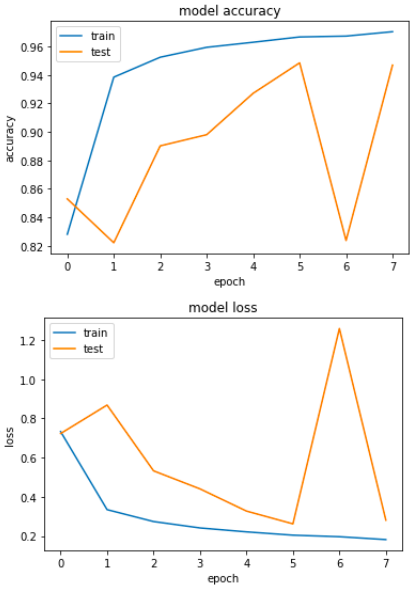
Training accuracy and loss for the 1-layer ResNet.

As we can see from the above plots, the results obtained on the test set are a bit unstable. Due to a rather long training time, we've trained the model only for 11 epochs. It is possible that the accuracy and loss on the test set would have had stabilize if we had used more epochs. On the other hand we can also see that the model's accuracy on the training data constantly increases so when training for more epochs we could observe a serious overfit.



Training accuracy and loss for the 2-layer ResNet.

For the 2-layer ResNet the accuracy and loss calculated on the test set seem to be a bit more stable than for the 1-layer ResNet but still there is a room for improvement.

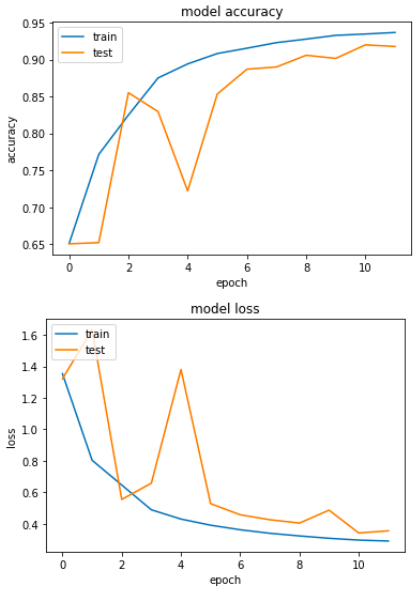


Training accuracy and loss for the 3-layer ResNet.

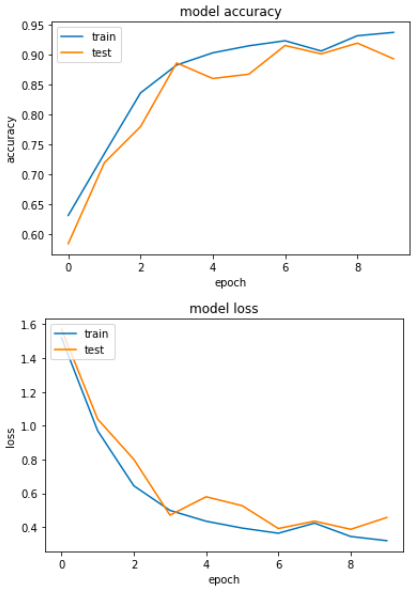
Surprisingly, the deepest, 3-layer model turned out to perform the worst.

4.3.2 Data augmentation

It turned out that it is not an easy task to find data augmentation that would improve the performance of our models. From the below plots it would seem that the performance of the model on the Kaggle's test set would be better than for the models that were trained without data augmentation.



Training accuracy and loss for the 2-layer ResNet with data augmentation.



Training accuracy and loss for the 3-layer ResNet with data augmentation.

We can see that both accuracy and loss curves stabilize quickly, especially for a 3-layer model. There's no sign of overfit, the accuracy is really high on the test set. It turned out though, that the models trained with data augmentation perform much worse (around 10%) on the Kaggle's test set than the models trained without data augmentation.

5 Final Kaggle score

The final prediction was made using an ensemble of 3 classifiers: 1-, 2- and 3-layer ResNets. We decided to calculate the mode of the 3 classifiers. If for a given observation the result was unambiguous i.e. the three classifiers predicted 3 different labels, such observation obtained the label predicted by 2-layer resnet since it had the highest score out of the forementioned models. As a result we obtained the following score on Kaggle:

Name	Submitted	Wait time	Execution time	Score
wyniki_komitet.csv	just now	1 seconds	1 seconds	0.76260

Complete

which is almost 1.5% higher than the best score of a single model the ensemble was build from.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[2] *Pathmind LSTM article*. https://wiki.pathmind.com/lstm#long.

[3] *TensorFlow Speech Recognition Challenge*. https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/.